Methods for Probabilistic Fault Diagnosis: An Electrical Power System Case Study

Brian W. Ricks [1,2], Ole J. Mengshoel [3]

[1] *University of Texas at Dallas, Dallas, TX 75080 USA*
[2] *USRP, NASA Ames Research Center, Moffett Field, CA 80523 USA*

bwr031000@utdallas.edu

[3] *Carnegie Mellon Silicon Valley, NASA Ames Research Center, Moffett Field, CA 80523 USA*

Ole.J.Mengshoel@nasa.gov

ABSTRACT

Health management systems that more accurately and quickly diagnose faults that may occur in different technical systems on-board a vehicle will play a key role in the success of future NASA missions. We discuss in this paper the diagnosis of abrupt continuous (or parametric) faults within the context of probabilistic graphical models, more specifically Bayesian networks that are compiled to arithmetic circuits. This paper extends our previous research, within the same probabilistic setting, on diagnosis of abrupt discrete faults. Our approach and diagnostic algorithm ProDiagnose are domain-independent; however we use an electrical power system testbed called ADAPT as a case study. In one set of ADAPT experiments, performed as part of the 2009 Diagnostic Challenge, our system turned out to have the best performance among all competitors. In a second set of experiments, we show how we have recently further significantly improved the performance of the probabilistic model of ADAPT. While these experiments are obtained for an electrical power system testbed, we believe they can easily be transitioned to real-world systems, thus promising to increase the success of future NASA missions.

1 INTRODUCTION

Due to inherent uncertainties in systems as well as in sensors, probabilistic methods are starting to play an important role in system health management. In diagnostics, there are different probabilities of failure for different types of components. There may also be sensor noise.

Fortunately, much progress has recently been made in (i) modeling systems, under conditions of uncertainty, using probabilistic graphical models, and (ii) performing diagnosis by means of such models (Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Olesen, 1993; Darwiche, 2000). Roughly speaking, these models admit relatively sparse, and hence computationally efficient, representation of conditional dependence and independence relationships of large multivariate probability distributions. In this paper, we focus on graphical models in the form of Bayesian networks and arithmetic circuits, and illustrate novel methods for failure diagnosis of hybrid systems using an electrical power system case study.

Systems that we want to diagnose, including electrical power systems (EPSs), are often hybrid, in that they exhibit both continuous and discrete behavior. Examples of continuous behavior, in the EPS setting, include (measurements of) voltage, current, and temperature, while discrete behavior is induced by protection and control devices, such as relays, circuit breakers, and sensors for such devices. In addition to the hybrid nature of many systems that are in need of advanced system health monitoring, they may also be partly event-driven, partly periodic and have non-trivial and varying system dynamics, including transients, associated with them. An example of events in the EPS context are user commands, while sampling of EPS sensors is an example of periodic behavior.

There is an urgent need for methods that bridge the gap between complex systems (such as electrical power systems) that are hybrid and may also exhibit some of the other issues (event-driven/periodic and varying dynamics) discussed above. Most existing diagnostic technologies typically have a discrete or continuous foundation, and diagnostics in a

hybrid, complex setting is an important topic for on-going research. In this paper, we develop methods for hybrid diagnosis by means of discrete probabilistic models (Bayesian networks and arithmetic circuits), and specifically develop novel techniques for handling abrupt continuous (or parametric) faults such as continuous stuck faults and continuous offset faults. The challenge associated with continuous stuck faults is that they may be easily confused with measurements that are normal but have a low level of noise. In particular, this happens if the sensor discretization level is of the same order of magnitude as the noise level from some sensors in a system. The challenge associated with continuous offset faults is that they are associated with small and continuous anomalies that are hard to detect using a discrete model. Specifically, since the magnitude of a continuous off-set is not known ahead of time, it is in practice impossible to discretize according to all possible off-set faults. The challenge associated with dynamics including transients is to avoid false positives.

In this paper, we discuss our contributions in the context of the ProDiagnose algorithm. ProDiagnose processes all incoming environment data (observations from a system being diagnosed), and acts as a gateway to a probabilistic inference engine. The inference engine analyzes the observations given to it by ProDiagnose, and computes diagnoses. ProDiagnose currently uses the Arithmetic Circuit Evaluator (ACE). ACE uses arithmetic circuits (ACs), which are compiled from Bayesian network models (Chavira and Darwiche 2007; Darwiche 2003). The primary advantage to using ACs is speed, which is key in resource-bounded systems such as aircraft and spacecraft (Mengshoel 2007). Given an appropriate probabilistic model, ProDiagnose diagnoses different types of faults for sensors and components at a high level of performance and accuracy. In this paper we focus on the introduction, in a Bayesian network, of discrete Change nodes, Delta nodes, and Stuck nodes, and how they are coupled with algorithms that process continuous data, and how the overall system provides high-performance diagnosis in the context of abrupt continuous faults, specifically continuous stuck faults and continuous offset faults.

The rest of the paper is structured as follows. In Section 2, we present work related to our research. Section 3 presents in more detail faults that may occur in EPSs such as ADAPT, and specifically discuss continuous faults that are non-trivial to handle in a discrete BN. In Section 4, we give an overview of the diagnostic process, including terminology and notations. In Section 5 we present the ProDiagnose algorithm. Section 6 is devoted to a discussion of the Bayesian network model structures, including how the ADAPT BN model is used for diagnosis of continuous stuck faults, continuous offset faults, and handling of transients. Section 7 gives an electrical power system case study, diving more in-depth into the ADAPT EPS. Section 8 presents the DXC framework and discusses experimental results from ProDiagnose, including the DXC-09 competition and latest experimental results using a newer BN model. In Section 9 we conclude and sketch future research directions.

2 RELATED WORK

We use in this paper Bayesian networks (BNs) to represent probabilistic multi-variate models (Lauritzen and Spiegelhalter 1988; Pearl 1988). A BN is a directed acyclic graph (DAG), combined with an associated set of conditional probability tables (CPTs). Each vertex of the graph represents a *discrete random variable,* represented visually as a *node*. Each node has a CPT of size that is dependent on the number of parent vertices, and the number of discrete *states* that these vertices contain. The directed edges typically represent the causal dependencies between variables. By *clamping* random variables (nodes), it is possible to compute the marginal probability of other vertices in the BN. The marginal probabilities can then be used to diagnose the system itself.

We currently use arithmetic circuit evaluation for probabilistic inference (and ACE as the inference engine). Arithmetic circuits are a fast way to evaluate Bayesian networks. An arithmetic circuit derives marginal probabilities by addition and multiplication operations (Chavira & Darwiche 2007; Darwiche 2003). During each ProDiagnose call to ACE, the partial derivatives of this AC are computed with respect to each discrete random variable. ProDiagnose queries the arithmetic circuit to return the marginal probabilities in constant time.

We identify two areas of related work on hybrid systems: research using Bayesian networks, and research using other techniques. Of particular interest is fault diagnosis in terrestrial and vehicular electrical power systems.

Among research using other techniques, we consider first model-based fault diagnosis in hybrid systems (Narasimhan and Biswas 2007; Daigle et al., 2008). Narasimhan and Biswas discuss a model-based diagnosis approach based on hybrid bond graphs (Narasimhan and Biswas, 2007). The approach integrates tracking (using an extended Kalman filter, and fault detection), fault detection (which compares estimated and observed signals), fault isolation, and fault identification. Successful experimental results are shown for a fuel-transfer system of a fighter aircraft. The work by Daigle et al. is also based on hybrid bond graphs (Daigle et al., 2008). This research is similar to ours in its emphasis on electrical power systems and ADAPT specifically; it also deals with abrupt continuous and discrete faults. Unlike our research, this work makes the single fault assumption (Daigle et al., 2008). RODON, a model-based approach (Karin et al., 2006) based on the general diagnostic engine (de Kleer and Williams, 1987), also participated in DXC-09 with good results (Bunus et al., 2009; Kurtoglu et al., 2009a; Kurtoglu et al., 2009b).

An optimization-based approach to fault diagnosis has been applied to ADAPT as well (Gorinevsky et al., 2009). This approach, which obtained good results in DXC-09, amounts to developing a linear model of the EPS circuit, and diagnosis is then based on solving a convex problem that includes faults and other hidden states.

We now turn to research using Bayesian networks. Based on the types of random variables they contain, we can partition BNs into three classes: discrete BNs, which contain discrete random variables only, continuous BNs, which contain continuous random variables only, and hybrid BNs, which contain both discrete and continuous random variables.

Based on clique tree propagation (Spiegelhalter and Lauritzen, 1988), Olesen developed an approach to exactly compute marginals in clique trees that are compiled from hybrid BNs (Olesen, 1993). In order to maintain exactness, the hybrid BNs were restricted to ones in which the continuous nodes are Gaussian and do not have discrete parents. For a continuous node, each discrete configuration of parents gives a linear Gaussian distribution. For each configuration of all discrete nodes, the continuous distribution is multivariate Gaussian, and this approach therefore is a generalization of mixtures of Gaussians.

Koller and Lerner also investigated hybrid BNs, and introduced an inexact particle filtering approach for computing marginals (Koller and Lerner, 2000). Each particle is an instantiation of non-evidence nodes X(t), and the belief state at time t is approximated by all particles. This particle filtering algorithm approximates marginals at time t over all non-evidence continuous and discrete nodes.

Even though it is natural to use hybrid BNs in hybrid domains such as EPSs, there are also limitations associated with doing so. In particular, the mathematics of hybrid BNs is non-trivial, thus the need to introduce restrictions (Olesen, 1993) or resort to approximations (Koller and Lerner, 2000). Arithmetic circuits, to which we compile our BNs, do not currently support continuous or hybrid BNs. Consequently, we discuss in this work hybrid methods, in particular diagnosis of abrupt discrete and continuous (or parametric) faults, which use discrete BNs. Discrete BNs have previously been used for fault diagnosis in terrestrial EPSs (Yongli et al., 2006; Chien et al., 2002), although not for the type of abrupt continuous faults that we stress here.

In this discussion of diagnosis of abrupt continuous faults, our main emphasis is on algorithms that discretize continuous signals, and in particular create from them discrete outputs that are used as evidence in discrete BNs. This extends our previous work, where we considered abrupt discrete faults (Mengshoel et al., 2008; Mengshoel et al., 2009); we also go into more technical detail than our previous paper on ProDiagnose (Ricks and Mengshoel, 2009). For a more detailed comparison of ProDiagnose to other systems in the ADAPT setting, we refer to discussions of the benchmarking framework and its application in DXC-09 (Kurtoglu et al., 2009a; Kurtoglu et al., 2009b).

3 BEHAVIORS AND FAULTS

There are many dimensions along which systems faults, such as faults in electrical power systems, may vary. Diagnostic techniques and systems typically vary accordingly, since it is very difficult to develop approaches that are able to detect or isolate along all of these dimensions equally well and with equal computational efficiency. We now discuss a few of these dimensions.

One dimension is speed of fault progression, where we distinguish between faults that progress very quickly (abrupt faults) versus faults that progress very slowly (incipient faults). Another dimension is fault persistency, where one typically distinguishes between persistent and intermittent faults. A third dimension is the fault type, where it is fruitful to distinguish between continuous (or parametric) faults and discrete faults. As an example, "stuck high" is a discrete fault, while "stuck at X", where X is a parameter that can vary over a real-valued interval, is considered a continuous fault. A fourth dimension is independent faults versus dependent faults; common cause faults and cascading faults are examples of dependent faults.

In this paper, we are concerned with abrupt continuous (or parametric) faults that are independent and persistent.[1] This contrasts with earlier work (Mengshoel et al. 2008; Mengshoel et al. 2009) where we also investigated independent and persistent faults, but they were abrupt and discrete. In this section we will use the ADAPT EPS to illustrate our approach.

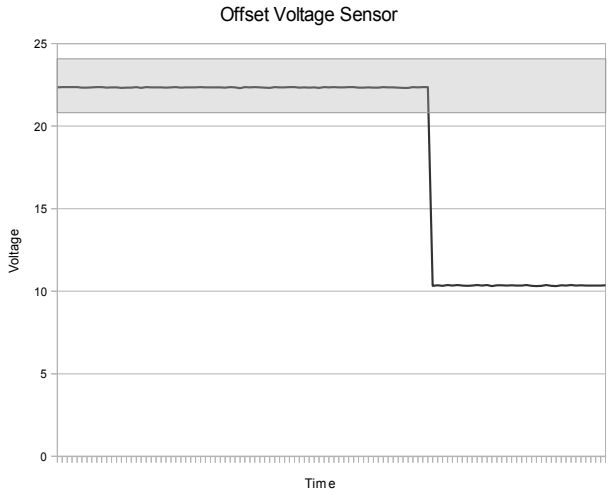

Figure 1: Graph showing the sensor readings of a voltage sensor over time. The voltage drop illustrates an offset sensor fault, a type of abrupt fault. The grey area represents the nominal range considered normal or healthy for this sensor.

In Figure 1, we see an example of sensor readings from a voltage sensor. Assuming this part of the EPS has power, the grey box represents the nominal range that this sensor is allowed to be in (to be considered healthy). This nominal range is dependent on other sensors within the EPS, meaning that other changes within the electrical power system itself may change what the nominal range for this sensor is at any time. In this scenario, the sensor's value (voltage) suddenly

[1]This is not to say that ProDiagnose cannot handle other fault types - we are currently investigating intermittent as well as cascading faults

drops downwards out of this range. The nominal range itself is still deemed to be the same as before the voltage drop however, and the sensor is now considered to be offset. This example (Figure 1) illustrates an offset of about -12 V, which is enough to throw the sensor out of its nominal range.

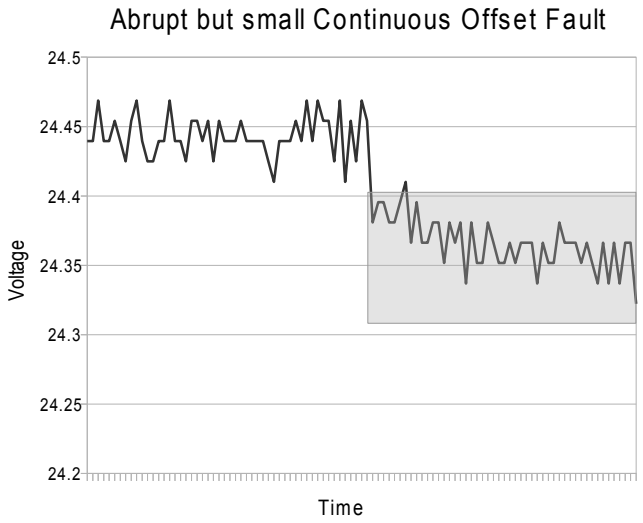

Figure 2: Graph showing the behavior of a degraded battery according to the closest voltage sensor downstream. The grey area represents the region in which the voltage has dropped very slightly due to a short.

Figure 2 illustrates an abrupt continuous offset fault of very small magnitude. These types of faults often cannot be diagnosed as quickly as the previous example (Figure 1). The graph shows a tenth of a volt drop of a power source, in this example a battery after a short circuit somewhere in the EPS. When factoring in the sensor noise, detection of this drop would be very difficult by using a nominal range due to the narrow scope of the range factored in with the noise spike both before and after degradation starts.

ProDiagnose handles this tiny abrupt fault by using weighted Cumulative Sums, or CUSUMs, to monitor the long term change in behavior of the sensor. The difference between the weighted average of the sensor's readings and the current sensor reading is added to the current CUSUM (which initially is zero). This pattern repeats for each sensor reading received, and thus the CUSUM is keeping a record of the overall trends (long-term behavior) of this sensor. This technique can exploit even minute changes in a sensor's behavior over a given time period.

If we were to let this fault play out over time, the battery would slowly exhaust itself, and the voltage would start to drop very gradually as a result. ProDiagnose would still use the same technique to catch it. In fact, if this scenario were to happen, ProDiagnose would simply continue to diagnose the same abnormality both after the initial voltage drop and during the degradation itself. If there is no initial voltage drop, this would be an incipient fault, but ProDiagnose would still catch it using CUSUMs. The detection time would depend in how gradual the degradation was. This technique would allow the fault to be diagnosed well before the degradation dropped below the nominal range.

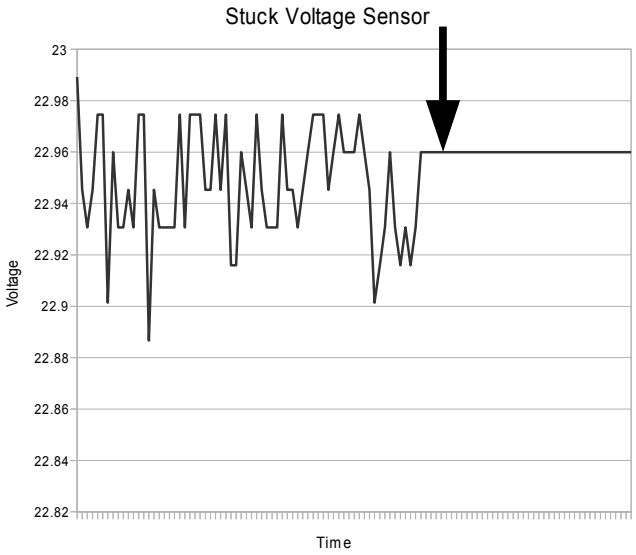

Figure 3: Graph showing the sensor readings of a voltage sensor over time. The graph after the arrow indicates the area in which this sensor has become stuck at the same value (a stuck fault).

Figure 3 shows a very common abrupt continuous fault. The noise associated with this sensor's readings are shown before the arrow. The sensor in the figure is moderately noisy, but can have short periods in which it returns the same sensor reading in subsequent samples, which is evident from Figure 3. After the arrow, all noise ceases to exist in the sensor, which is not characteristic for this type of sensor. ProDiagnose cannot detect the fault immediately due to ambiguity between the initial fault and normal periods of equal-valued sensor readings. However, after a certain interval of time with no change in sensor readings, the sensor is then diagnosed as operating abnormally. This span varies depending on nominal characteristics between different sensor types.

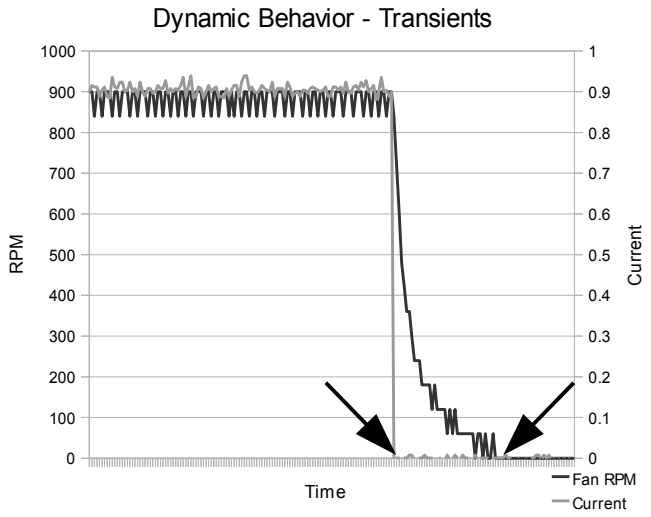

Figure 4: Graph showing the behavior of a failed fan according to its RPM sensor and a current sensor in series with the fan itself. The area between the arrows shows the difference in behavior between current and fan blade RPM immediately after the failure.

Figure 4 illustrates a fan failure in an EPS. Power is applied throughout this time interval, but the fan suddenly stops working, and both the RPM of the fan blades and the current draw of the fan drop to zero. If the sensors in the EPS that directly monitor the fan (RPM sensor) or indirectly (current sensors in series with the fan) are all deemed to be healthy, and there are no other component failures upstream from the fan itself, then this fan would be considered failed.

Notice the area of the graph between the arrows. The current drops immediately to zero following the failure, which indicates an abrupt fault. However the fan blades do not immediately stop spinning, but rather spin down gradually. This characteristic indicates dynamic transient behavior. During this time frame, if only taking into consideration the RPM values themselves, it would appear that perhaps this speed sensor is abnormal as it is not reading zero. In fact, until this speed sensor hits the nominal range considered to be zero, the sensor will most likely be diagnosed incorrectly as offset. To avert this type of problem, more evidence is introduced to specify the short-term behavior of the fan blades at a given RPM reading. If the RPM is dropping over this period of time, then this evidence combined with what is known from the current sensor along with the RPM reading itself can lead to a more accurate diagnosis.

While this type of fault deviates from the main emphasis on abrupt continuous faults, we include this example to illustrate the technique used to diagnose it. ProDiagnose monitors this type of short term behavior by taking the difference of the current and previous weighted sensor averages. We call this a *Delta*. Unlike with CUSUMs, no record of the sensor's long term behavior is recorded. This technique is used to give a good picture of the short-term behavior of a sensor. In the example above, the decreasing fan blade RPM would result in a consistent negative delta until the fan blades stopped spinning.

4 OVERVIEW OF DIAGNOSTIC PROCESS

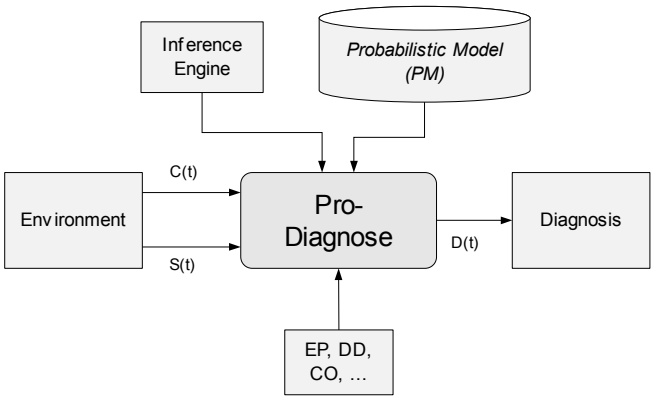

Figure 5: The ProDiagnose Architecture. Two types of diagnosis-related messages can be received, commands, C(t) and sensor readings, S(t) (or sensor data). Commands can be received any time, whereas sensor data comes in at specific times, according to the sample cycle. Diagnosis, D(t), is sent after each sample cycle completes.

Figure 5 depicts the ProDiagnose architecture. The environment provides our sensor data and commands from the system we are diagnosing. The probabilistic model provides a model of the system (to diagnose, such as a Bayesian network), which works with the inference engine to provide diagnoses each time ProDiagnose receives data from the environment.

How the inference engine receives the sensor and command data is dependent on various *parameters* ProDiagnose uses during the diagnostic process. These parameters, along with other notation and definitions, are introduced in Section 4.1

The fault examples presented in this paper are from the ADAPT EPS, a physical EPS developed by NASA. Sensor data obtained from experiments (scenarios) utilizing ADAPT are converted to a format specified by the evaluation framework ProDiagnose uses for scenario input (Kurtoglu et al., 2009a; Kurtoglu et al., 2009b). ProDiagnose uses these scenarios along with the ADAPT BN to diagnose faults in the system, which consist of abrupt continuous faults (see Section 3).

4.1 Notations and Definitions

The following is a list of all ProDiagnose parameters and their purpose:

Sample Cycle, t_{SC}: The amount of time, measured in milliseconds, between sample readings.

Command Epsilon, t_{EP}: A global threshold for determining if a given command should be clamped as evidence immediately or queued in regard to the time stamp of the last sensor set. This is discussed in more detail in the Command Data section (Section 5.3).

Diagnosis Delay, t_{DD}: A global value, measured in milliseconds, that gives the delay to start diagnosis output. Diagnosis delay is used at the beginning of environment monitoring. This variable is useful to filter out transients and other normal behavior that may appear abnormal and thus have false positive diagnoses associated with them.

Command Offset, t_{CO}: A global value, measured in milliseconds, that gives the delay to output diagnosis when a command is received. This variable is useful for situations in which, for some *n* millseconds after a command is issued to a component, transients from sensors cause ProDiagnose to produce false diagnosis. Depending on when the command was issued in regard to the next sensor sample set, and also the length of the transient, t_{EP} may not in itself be able to prevent the false diagnosis.

Sensor Stuck Delay, t_{SSD}: A value, measured in milliseconds, that gives, for a sensor with a reading that is the same, a maximum amount of time to wait before setting that sensor to a stuck state.

Other notation and definitions are described as follows:

PM (Probabilistic Model): The probabilistic model represents the system that ProDiagnose will diagnose. The probabilistic model that ProDiagnose uses is an Arithmetic Circuit compiled from a Bayesian Network.

e (evidence): e represent the evidence that is used in the diagnosis process. Evidence comes from commands and sensor readings (the environment, Figure 5).

4.2 Diagnostic Scenario using ProADAPT

Using a scenario from the DXC-09 Industrial Track Tier 2 as an example, we illustrate the importance of the ProDiagnose parameters in accurate fault detection. We will cover specifics of the ADAPT EPS in Section 7.

The example scenario follows two current sensors in the ADAPT EPS, IT261 and IT281. IT261 is located just before an inverter (Figure 18, 'it' sensor closest on left to bottom 'INV') and monitors current flow into the inverter and DC load bank. IT281, located just before the a DC load bank (Figure 18, 'it' sensor closest to bottom load bank with 'DC' load) only monitors current flow into the DC load bank. The following graphs show the current readings of these sensors over a specific period of time. The arrows indicate important areas of the graphs that will be discussed shortly.

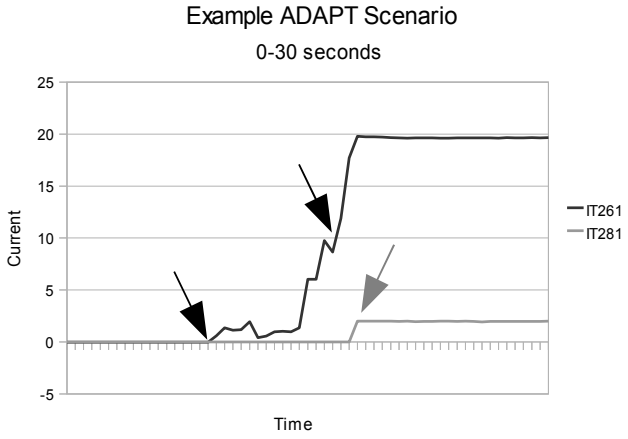

Figure 6: The first 30 seconds of an example ADAPT scenario, following two current sensors.

Starting out the scenario, all relays are open, and no current is flowing through any of the sensors. Relays start closing, powering up the EPS (18 relays close in this span of time), Looking at IT281 (Figure 6), we see a nice near instantaneous change in current flow at the grey arrow. Looking closely, there is a tiny transient here as the rate of change of the current flow is not instantaneous. t_{CO} or t_{EP} would work here to eliminate any false positives calculated. However, for IT261, the changes in flow of current are not nearly as quick or linear as for IT281, as indicate by the arrows. The inverter takes a bit of time to fully power up, and during this time the current flow changes frequently. This behavior is very unpredictable and hard to monitor. Thus, ProDiagnose uses t_{DD} to suppress any false positives generated during this time until after the EPS reaches a steady state.

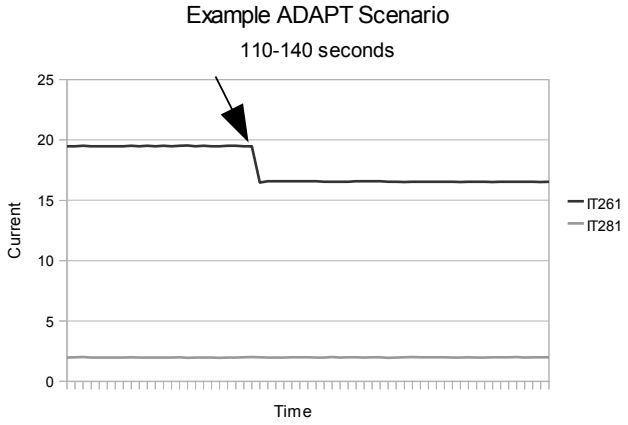

Figure 7: A span of 30 seconds starting 110 seconds into the example ADAPT scenario.

At about 110 seconds into the scenario, a relay in the second AC load bank opens (represented by the arrow in Figure 7), cutting current to a light bulb connected to it. This results in a current drop for sensor IT261. IT281 however monitors the DC bank and is unaffected by the relay. While the drop

in current for IT261 is near instantaneous, it is still possible for false positives. If the command to open the relay was given right before the next sensor sample set, it is very possible that the next sensor reading itself will still be in the same threshold range (as the current could still be dropping, notice that the drop in Figure 7 is not totally vertical). This may be diagnosed as offset, depending on the magnitude of the current drop. Commands given farther from the next sample set will most likely not be affected, as by this time (say 100ms) the current drop should have completed easily. To avoid these issues, parameters t_{EP} or t_{CO} can be used, either by queuing the command and waiting for the transient to pass (t_{EP}), or suppressing diagnosis until the transient passes (t_{CO})

5 DIAGNOSTIC ALGORITHM

Before we dive into the algorithms, we introduce notations and definitions:

5.1 Notations and Definitions

We now describe the data structures associated with ProDiagnose. These structures store information related to the current state of the system being diagnosed. A node represents one specific instance in a set.

C (Command Set): A Command node $C \in \mathbf{C}$ represents a command given to a component. An example would be a command to open or close a relay. This is shown visually as C(t), Figure 5.

S (Sensor Set): A Sensor node $S \in \mathbf{S}$ represents the current reading of a sensor. This reading is discretized from real-valued S(t) in Figure 5. which represents a range for real-valued sensors, or the actual state of 0 or 1 for a boolean position sensor. The discretized sensor reading is clamped as evidence in the BN.

H (Health Set): A Health node $H \in \mathbf{H}$ represents the current health state of a component. The set of states of a node H is partitioned into normal and abnormal states. Abnormal states indicate a fault in the component. Any component or sensor that is not healthy is output by ProDiagnose as faulty (with the faulty state being the abnormal health state). In Figure 5, D(t) represents health nodes with abnormal states.

ST (Stuck Set): A Stuck node $ST \in \mathbf{ST}$ represents the stuck state of a sensor. A sensor becomes stuck when its reading is the same over a period of time, regardless of what the underlying process state is.

D (Delta Set): A Delta node $D \in \mathbf{D}$ represents the difference (delta) between the current sensor reading $S(t)$ and its previous reading $S(t-1)$:

$$\Delta S = S(t) - S(t-1)$$

The discretization of D into three states corresponds to the following three cases: $\Delta S < 0$, $\Delta S = 0$, and $\Delta S > 0$. Note that **D** is not the same as D(t) in Figure 5.

CH (Change Set): A Change node $CH \in \mathbf{CH}$ represents overall trends in sensor readings (long term behavior), using CUSUMs:

$$CUSUM(t) = (S(t) - \{S(t)_{WEIGHTED} + \ldots + S(t-p)_{WEIGHTED}\}) + CUSUM(t-1)$$

$S(t)$ represents the current sensor reading, which is subtracted from the weighted average of a contiguous subsequence of sensor readings, from the current reading ($S(t)_{WEIGHTED}$) to the sensor reading p cycles back ($S(t-p)_{WEIGHTED}$). This difference is then added to the previous CUSUM value ($CUSUM(t-1)$). *CH* nodes are good for detecting small changes in sensor readings over a period of time. This change is clamped as evidence, but it also depends on H, as certain states of health for relevant components can play a role in how the change nodes affect the rest of the BN.

A (Attribute): An Attribute $A \in \mathbf{A}$ represents a subset of nodes that describe various attributes of a component. These attributes could be voltage V and current I for an electrical device. A usually depends on $A' \in \mathbf{A}$ upstream, where $A' \neq A$.

CL (Component State): A Component State node $CL \in \mathbf{CL}$ represents a generalized state of operation for the component.

Base_Component: A *base_component* represents a physical component of a system in the probabilistic model. base_components are used as a common link for lookups of various parts that all share the same *base_component*. For example, in an EPS, a fan component may have a sensor that monitors blade RPM. This sensor's *base_component* would be the fan.

The ProDiagnose algorithm can be broken down into two stages: The pre-processing and diagnosing stages. The pre-processing stage initializes ProDiagnose to a state in which it can start accepting data from an environment. The diagnosis stage is executed each time data from the environment is received.

5.2 Pre-Processing Stage

```
1  Algorithm ProDiagnose(t_EP, t_DD, t_CO)
2  Begin:
3    initialize_DA(t_EP, t_DD, t_CO, Init_Params : PDB)
4
5    Send_Message(Message : M = DA_Ready)
6
7    do
8    Begin:
9      receive Message : M from environment
10
11     Process_Message(M, t_EP, t_DD, t_CO)
12   loop until M = Terminate
13 End
```

The pre-processing stage sets up ProDiagnose, including parameters and all data structures that will be used during diagnosing.

5.3 Diagnosis Stage

The diagnosing stage analyzes each *message* **S**(t) or C(t) when they come in and outputs diagnosis of abnormal health *(H)* states according to the sample cycle, t_{SC}. The first course of action is to determine the data type of the incoming message, which can be either a sensor message, command message, or termination message. ProDiagnose evaluates the *PM* and computes diagnoses only when sensor data is received.

```
1  Algorithm Process_Message(Message : M, t_EP, t_DD, t_CO)
2  Begin:
3    if M = Scenario_Status : Terminate then Exit
4
5    if M = C(t) : (Command : C_Command, Value : V)
6    Begin:
7      C ← get_node(C_Command)
8      t_i ← C.timestamp
9      t_j ← S(t - 1).timestamp + t_SC
10     if t_j -  t_i < t_EP
11       command_queue ← C
12     else
13       C.command ← Discretize_For_PM(V)
14   End if
15
16   if M = S(t)
17   Begin:
18     for each S(t) : (Sensor : S_Sens, Value : V) ∈ S(t)
19     Begin:
20       S ← get_node(S_Sens)
21       S.value ← Discretize_For_PM(V)
22
23       if D ∈ Base_Component(S)
24         D.value ← Discretize_For_PM(Calc_Delta(D))
25
26       if ST ∈ Base_Component(S)
27         ST.value ← Discretize_For_PM(Calc_Stuck(ST))
28     End for
29
30     for each CH
31       CH.value ← Calc_Change(CH)
32
33   End if
34
35   Calculate_Marginals(PM)
36
37   Output_Diagnosis(H, t_DD, t_CO)
38
39   Update_Command_Queue(command_queue)
40 End
```

Scenario_Status (Line 3): This datatype is a constant specifying any status updates that arrive to ProDiagnose as message M. If M is the constant specifying termination, then ProDiagnose frees up its resources and exits gracefully.

C(t) (Line 5): This datatype is a tuple, (*C_Command, V*), in which *C_Command* is a command given, and V is the value of the command. ProDiagnose first fetches the appropriate *C* node (line 7). It then checks the timestamp of the command. If the command $C(t_i)$ has come in too close to $\mathbf{S}(t_j)$, where $j > i$ and $t_j - t_i < t_{EP}$, then we queue the command (line 11). Otherwise we update the *C* node with the new command (see Figure 11). The queuing of commands is due to the following.

In a physical system, such as an EPS, there is often a significant delay between the time a command is issued and the time it is sensed to have taken effect. In addition comes communication delays, both to actuators and from sensors. ProDiagnose will queue commands to attempt to make sure that commands are only input to the *PM* after sensor readings from the system reflect the effects of these commands. In ADAPT, keeping commands queued for one sample period usually achieves the desired effect.

S(t) (Line 16): This datatype is a set, $\{(S\_Sens, V) \mid S\_Sens \in \mathbf{S}\}$, in which *S_Sens* is a sensor, and *V* is the value for the sensor. Each sample has a key/value pair for every sensor in the network. The keys map to an *S* node, and the values (*V*) represent the current sensor reading for the respective *S* node. For each *S* node, its new sensor reading is discretized (Line 21) and value updated to the new reading. During each iteration ProDiagnose also looks for any *D* or *ST* nodes that share the same *base_component* as the *S* node in the network. These operations consist of simple lookups using the *base_component* for the sensor.

If a *D* or *ST* node is found for a specific *base_component*, then its value is updated using the current sensor value (lines 24, 27). This value is further discretized for clamping as evidence in the network.

After all *S* nodes are processed, ProDiagnose updates the values of any *CH* nodes that may be present in the Bayesian network. Since the value of *CH* nodes can be derived from *any* sensor in the Bayesian network (as opposed to *D* and *ST* nodes in which the *base_component's* sensor value is always the one used), a reference to this bound *S* node is stored in the *CH* node. Because of this, we can iterate through the *CH* nodes (updating their values) after all *S* nodes are updated, as opposed to doing *CH* node lookups for each *S* node (though it is worth mentioning that *CH* nodes can be treated similar to *D* and *ST* nodes). At this point all our input nodes are ready for clamping to the network and evaluation of the network itself.

```
1  Algorithm Discretize_For_PM(Value : V, Thresholds : TH)
2  Begin:
3  A ← NEGATIVE_INFINITY
5
6    for each N ∈ TH
7    Begin:
8      B ← N
9      if V ≥ A and V < B
10       return TH.Index(N)
11     else
12     Begin:
13       A ← B
14     End else
15   End for
16
17   return TH.Index(TH.size + 1)
18 End
```

The Discretize_For_PM method (Process_Message, lines 21, 24, 27) takes the current sensor value and returns an index value that is used in network nodes as states (clamped evidence). This index is the index value between two *thresholds*. A threshold has **TH.**size + 1 different Index values (line 6) that are possible, starting at 0, where **TH.**size is defined as the number of thresholds N in the set **TH**. The discretized value is Index(N) for which V is [A, B) (lines 9, 10), or Index(**TH.**size + 1) if V is above all thresholds (line 17). For example, a sample sensor has three discrete states in the *PM*: low, mid and high, which correspond to index values 0, 1 and 2 respectively. Two sample thresholds are given: 50 and 100. Any sensor reading below 50 is given an

index of 0, [50, 100) is given an index of 1, and above 100 is given index 2.

The following algorithms perform dynamic processing in the Bayesian network:

```
1  Algorithm Calc_Delta(D)
2  Begin:
3    I ← Sensor_Average(Base_Component(D).S)
4    I_prev ← Sensor_Average(Base_Component(ST).S(t-1))
5    D.value ← I − I_prev
6
7    return D
8  End
```

```
1  Algorithm Sensor_Average(S)
2  Begin:
3    Sum ← 0
4
5    for each S  ∈ {S(t), ..., S(t − p)}
6    Begin:
7      Sum ← Sum + S.value * S.weight
8   End for
9
10   return A
```

The Calc_Delta method (Process_Message, line 24) returns the difference between the current and previous weighted averaged sensor values of the delta *D* node's *base_component* (lines 3, 4: Calc_Delta, Figure 9, see Section 5.1, **D**). The average is defined as the summation of any contiguous subsequence of sensor readings and their corresponding weights (line 7: Sensor_Average) from the current *S*(t) sample cycle to *S*(t – *p),* defined as the *p* sample cycles back in the timeline.

```
1  Algorithm Calc_Stuck(ST, Counter : I, Sensitivity : K)
2  Begin:
3    current_value ← Base_Component(ST).S.value
4    previous_value ← Base_Component(ST).S(t-1).value
5    J ← current_value − previous_value
6
7    if J = 0 and I ≥ K
8      return 0
9    else if J ≠ 0
10     I ← 0
11   else
12     I ← I + 1
13
14   return J
15 End
```

The Calc_Stuck method (Process_Message, line 27) analyses a component's sensor values for readings that are repeatedly identical, defined if J = 0, by subtracting the current *S*(t) and previous *S*(t - 1) values of the *ST* nodes' *base_component* sensor (line 5, Figures 8-10). Each time J = 0 a counter I is incremented. If this pattern continues past a given *sensitivity threshold* K so I ≥ K (line 7), the *ST* node is considered stuck. The pattern is broken if J ≠ 0 during a sample cycle (line 5), at which point I is reset to 0 (line 9). A stuck node *ST* has three discretized states, 0, 1, and 2, where 1 represents stuck, and 0, 2 represent non-stuck states.

```
1  Algorithm Calc_Change(CH, CUSUM : U)
2  Begin:
3    S ←  CH.Bound_Sensor
4    I ← Sensor_Average(S)
5    U_prev ← U
6    U ← (S.value − I) + U_prev
7
8    if U < CH.Lower_Threshold
9      return 0
10   else if U > CH.Upper_Threshold
11     return 2
12
13   return 1
14 End
```

The Calc_Change method (Process_Message, line 31) calculates a continuous CUSUM, or cumulative sum, which is used to detect slight changes, or *trends,* in a sensor reading over time. The current CUSUM U is calculated by taking the current sensor reading *S* from the *CH* nodes' bound sensor (Figure 10, see Section 5.1, **CH**) and subtracting it from an averaged sensor reading I (lines 4, 6), in the same way as for *D* nodes (see Sensor_Average algorithm). This difference is then added to the previous CUSUM, and updated as the new current CUSUM U (line 6). Very slight changes that form a trend will over time will cause the CUSUM to consistently increase or decrease. If this change accumulates to the point where the CUSUM's value to drop below a lower threshold (line 8) or above an upper threshold (line 10), the index of the *CH* node will change in the *PM* to 0 or 2, respectively.

```
1  Algorithm Calculate_Marginals(PM)
2  Begin:
3    for each Node : N ∈ {S,C,D,ST,CH}
4      e ← fetch_current_evidence(PM, N)
5
6    for each H ∈ H
7      H.state ← argmax(P(H | E = e))
8
9    return H
10 End
```

In the Calculate_Marginals method (Process_Message, line 35), ProDiagnose clamps as evidence all of the input nodes (lines 3, 4). Our probabilistic models will always have *S* nodes, but not necessarily *C, D, ST,* or *CH* nodes. ProDiagnose then calculates the marginals, P(*H* | **E** = ***e***), for all **H** (lines 6, 7). The output from the inference engine gives the DA the states of **H**. For each *H* ∈ **H**, ProDiagnose takes the most likely value for that node and assigns it as the new health state (line 7).

```
1  Algorithm Output_Diagnosis(H, t_DD, t_CO)
2  Begin:
3    Candidate Set : CS
4
5    if first execution of Algorithm
6      dd ← t_DD
7    if received C(t) within last sample cycle
8      co ← t_CO
9
10   if dd = 0 and co = 0
11   Begin:
12     for each H ∈ H
13     Begin:
14       if H.state = abnormal
15         CS ← H
16     End for
17   End if
18
19   if dd > 0
20     dd ← dd − 1
21   if co > 0
22     co ← co - 1
23
24   return CS
25 End
```

If the diagnosis delay has reached 0, *dd* = 0 (initially set during the first iteration of this algorithm), and there is no current command offset, *co* = 0 (line 10), ProDiagnose will output a four-tuple (*t*, CS, DS, IS) as D(*t*) (Figure 5, Process_Message, line 37) if any abnormal health states are detected. *t* is the current time, CS is a *candidate se*t, DS is a *boolean detection signal*, and IS is a *boolean isolation signal*. A candidate set CS is a set containing zero or more *candidates*. DS and IS are simply: DS = IS = (|CS| > 0). If CS is non-empty, we have CS = {C_1, ..., C_n}, where n ≥ 1, with each candidate C in CS consisting of two-tuples like

this: C = {(H_1, a_1), ..., (H_m, a_m)}, for m ≥ 1. For ProDiagnose, a health node H_i is included in a candidate C, along with a most likely state a_i, if and only if that state is abnormal. ProDiagnose always outputs exactly one (H_i, a_i) tuple per candidate, and thus candidate weights do not play a role (and have for simplicity been kept out of the discussion above). If *dd* > 0, then it decrements by 1 (line 20). This also happens with *co* > 0 (line 22). *co* will be set to its original value t_{CO} each time ProDiagnose receives a command within the last sample cycle of **S**(t).

```
1  Algorithm Update_Command_Queue(command_queue)
2  Begin:
3    for each C ∈ command_queue
4    Begin:
5      C ← get_node(C_Command)
6      t_i ← C.timestamp
7      t_{j+1} ← S(t).timestamp + t_SC
8
9      if t_{j+1} - t_i < t_EP
10       keep command in queue
11     else
12     Begin:
13       pop command from queue
14       C ← V
15     End else
16   End for
17 End
```

The last step taken by ProDiagnose after diagnosis output is updating the command queue (Process_Message, line 39), pulling any commands $C(t_i)$ whose timestamp is considered to be out of range of the next sample timestamp $\mathbf{S}(t_{j+1})$ according to the command epsilon, $t_{j+1} - t_i < t_{EP}$ (lines 8, 9).

6 BAYESIAN NETWORK (BN) STRUCTURES

The Bayesian networks ProDiagnose employs for EPSs have two types of *parts*: components and sensors. A component models a physical device in the EPS, such as a fan, circuit breaker, relay, or light bulb. A sensor models a physical sensor in the EPS. Sensors can take measurements of components or *wires*. In the ADAPT EPS for example, *e* (voltage) and *it* (current) sensors are wire sensors.

6.1 Component/Sensor structures

Components and sensors have specific structures within the Bayesian network. Figure 19 (Section 7.2) shows how these structures interconnect to form the entire system.

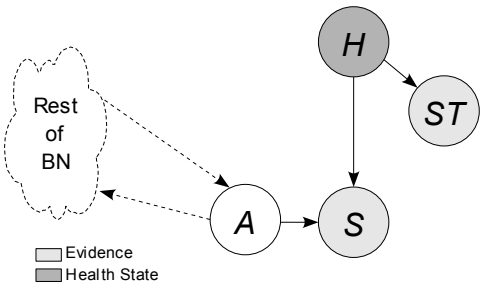

Figure 8: The Bayesian network representation of a basic sensor, such as a voltage, current, or frequency sensor. These types of sensors utilize stuck *ST* nodes for stuck fault diagnosis.

Figure 8 shows our BN representation of a sensor such as a voltage or current sensor. In the Bayesian network, these sensors are connected on wires from one component to another. The wire connection in the figure is represented as an Attribute node *A* (refer to section 5.1, data structures, for notations and definitions). These sensors are usually continuous-valued sensors, as indicated by the Stuck *ST* node present to provide stuck fault diagnosis within the Bayesian network (Section 5.3, Calc_Stuck method). This represents a basic structure for sensors in general, and many sensors take this form.

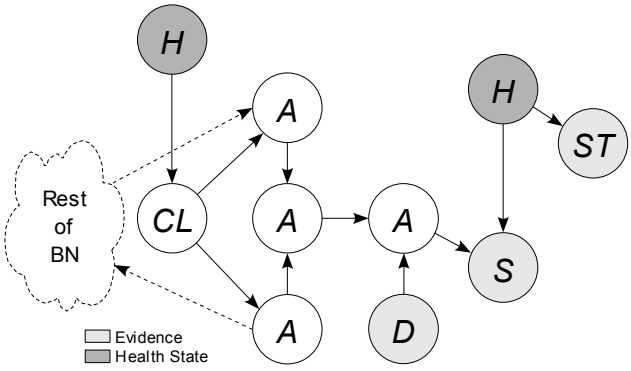

Figure 9: The Bayesian network representation of a component type with sensor such as a fan, pump or light bulb. These types of components utilize delta *D* and stuck *ST* nodes for their sensors.

Figure 9 takes our sensor representation from Figure 8 and adds to it a component to form our BN representation of a component plus sensor structure. This resulting structure is used for a component that has a sensor directly monitoring it. A physical example would be the fan and attached RPM sensor in the ADAPT EPS. The RPM sensor is represented in the Bayesian network as Figure 8 (the right section of Figure 9), and the fan itself (the component) is represented by the rest of Figure 9. These components may also utilize the Delta *D* node to provide the short term behavior of the component as evidence. This short term behavior is derived from the sensor *S* node's readings (Section 5.3, Calc_Delta method).

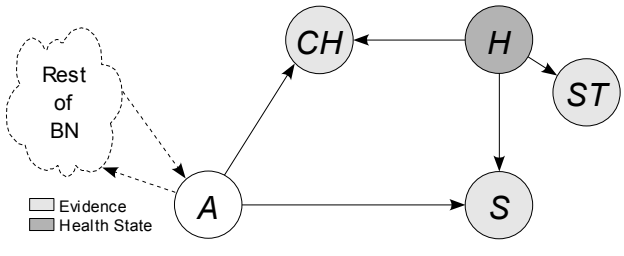

Figure 10: The Bayesian Network Representation of a bound sensor (source sensor) *S* to a change node *CH*. *CH* depends on both *A* and *H*.

Figure 10 represents the basic sensor structure from Figure 8 with one extra node added: a Change node *CH* to provide long term sensor behavior to components that have at the

very least an indirect relationship to the sensor itself. Using Figure 2 (Section 3) as an example, we would implement the *CH* node to provide long term behavior of the battery's voltage. The *CH* node would obtain its values from the CUSUM (Section 5.3, Calc_Change method) of a voltage sensor downstream of the battery (any voltage sensor who's readings are directly influenced by the battery could be used). This would be the *CH* node's bound sensor. In Figure 10, the sensor structure to the right would represent the voltage sensor, and the *A* node would represent part of the battery component's structure. Note that the battery does not have to be directly connected to the voltage sensor. In the ADAPT BN (Figure 18), for example, there are currently two nodes separating the battery component's structure from the closest in-line voltage sensor.

Another example is the new load bank monitoring model in the ADAPT Bayesian network. In this example, the *CH* node provides long term behavior of the current sensor monitoring the load bank to all the components within the load bank itself (in Figure 10, the load bank can be represented as the *A* node for simplicity). The CUSUM for this *CH* node is discretized into many states representing combinations of component failures within the load bank. This long term behavior is then used to pinpoint abrupt continuous faults within the load bank, based on this behavior along with other evidence from sensors within the bank.

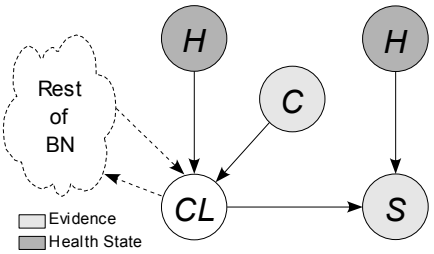

Figure 11: The BN representation of a component such as a relay or circuit breaker and its sensor. This type of component can be commandable via the *C* node.

Figure 11 shows the BN representation for a component, such as a relay or circuit breaker, and its corresponding sensor. Notice how the sensor does not incorporate a Stuck *ST* node. This is due to these sensors not giving real-valued readings. In the case of relays and circuit breakers, they can only be open or closed (so only two readings can be obtained from the sensors, either open or closed). Since circuit breakers tend to remain closed until tripped, it would appear over time that these circuit breakers were stuck in the closed state. In fact, if a circuit breaker or relay were to become stuck, it would mean here that the component or sensor was stuck in a state not considered healthy (closed if a relay was supposed to be open for example).

The Command *C* node introduces commands as evidence into the BN. For a relay, these commands will tell the relay to either open or close.

Associated with each node in a Bayesian network model is a *Conditional Probability Table* (CPT). The CPT gives the conditional probability that a specific node will be in a specific state given the state values of its parent nodes.

H	
healthy	0.85
offsetToZero	0.02
offsetToLow, offsetToMid, or offsetToHigh	0.04
stuck	0.01

Table 1: The CPT for a health node *H*. This CPT represents the health of a fan RPM sensor. States with the same conditional probability are grouped together for easier reading of the tables.

S					
H	*A*	**zero**	**low**	**mid**	**high**
healthy	**zero**	0.997	0.001	0.001	0.001
	low	0.001	0.997	0.001	0.001
	mid	0.001	0.001	0.997	0.001
	high	0.001	0.001	0.001	0.997
offsetToZero	**zero, low, mid, or high**	0.997	0.001	0.001	0.001
offsetToLow	**zero, low, mid, or high**	0.001	0.997	0.001	0.001
offsetToMid	**zero, low, mid, or high**	0.001	0.001	0.997	0.001
offsetToHigh	**zero, low, mid, or high**	0.001	0.001	0.001	0.997
stuck	**zero, low, mid or high**	0.001	0.333	0.333	0.333

Table 2: The CPT for a sensor node *S*. This CPT represents a fan sensor. Sensor readings are after *discretization* clamped to *S* nodes.

As mentioned in the notation in Section 5.1, a health node *H* gives the health state of a component or sensor in the Bayesian network. Most *H* nodes follow the CPT pattern shown in Table 1. Sensor *S* nodes represent sensors, and are evidence nodes in the Bayesian network. Sensor readings are clamped to *S* nodes as evidence. Evidence nodes are the way in which ProDiagnose inputs information to the Bayesian network.

ST			
H	**negDelta**	**zeroDelta**	**posDelta**
healthy	0.499	0.002	0.499
offsetToZero, offsetToLow, offsetToMid, or offsetToHigh	0.499	0.002	0.499
stuck	0.001	0.998	0.001

Table 3: The CPT for a stuck node *ST*. Stuck nodes tend to have the same CPT pattern. This CPT represents the stuck state of a fan sensor.

Stuck nodes *ST* are used to make a stuck state more probable within the same sensor's health node *H*. The two *ST* states *negDelta* and *posDelta* refer to a negative or positive change, respectively, in the sensor *S* node's sensor reading. The *zeroDelta* state represents a stuck state. After the *SSD* (t_{SSD}) has been reached, this state will be clamped in the *ST* node. When an *ST* node is clamped to *zeroDelta*, the *H* node's state has a very high probability (99.8%, Table 3) of being stuck, and since the *ST* node is directly connected to it, it yields great influence over the most likely value of the *H* node. We have equal conditional probabilities for the stuck state in the *S* node, Table 2, to make sure that the *S* node itself cannot yield any considerable influence on the *H* node being stuck. The one exception is when the sensor state is zero (Table 2). The very low conditional probability here for the stuck state prevents false stuck faults for sensors that can be nominally reading zero continuously, such as RPM sensors (the example CPT shown in Table 2).

CH				
H	*A*	**low**	**nominal**	**high**
healthy	**zero**	0.998	0.001	0.001
	low	0.998	0.001	0.001
	mid	0.001	0.998	0.001
	high	0.001	0.001	0.998
OffsetToLo, offsetToHi, or offsetToMax	**zero, low, mid, or high**	0.333	0.333	0.333
stuck	**zero, low, mid or high**	0.333	0.333	0.333

Table 4: The CPT for a change node *CH*. This CPT represents the general layout of *CH* nodes. The conditional probabilities for all non-healthy states of the *H* node will always be equal.

Change nodes *CH* are used to provide extra evidence for components in the BN that have states which cannot be properly determined by other evidence alone (similar to Delta *D* nodes). An example is trying to pinpoint a component failure in a bank of components, using only a single current sensor that monitors the current flow entering the entire bank. A *CH* node can give us extra evidence related to how much the current sensor changes during a component failure in the bank using CUSUMs.

A *CH* node will always be in a *nominal* state when it's CUSUM is near zero, implying no distinct long-term behavioral changes. In Table 4, the *CH* node has two other states, *low* and *high*. It is worth noting that *CH* nodes can have as many states as needed (in the load bank example stated above, the *CH* node would need one state for each component in the bank).

6.2 The Bayesian Network in Action

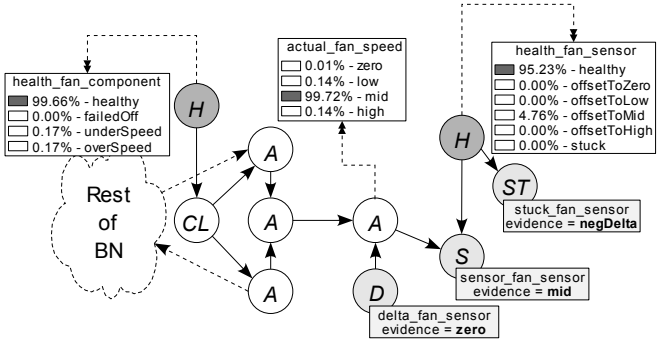

Figure 12: The marginal distributions for health *H* nodes health_fan_component and health_fan_sensor as well as the actual_fan_speed attribute *A* node (same representation as in Figure 9, a fan and its sensor). The actual_fan_speed *A* node represents the actual state of the fan's blades.

Figure 12 represents the current states for select nodes of a fan component and sensor (same structure as Figure 9) in an example scenario. We see the most likely values for the health *H* nodes of a fan component and sensor, based on the evidence shown (Figure 12). Despite being suppressed here to save space, the rest of the Bayesian network also influences these outcomes. Notice how the *actual_fan_speed A* node agrees with the evidence of the *S* node.

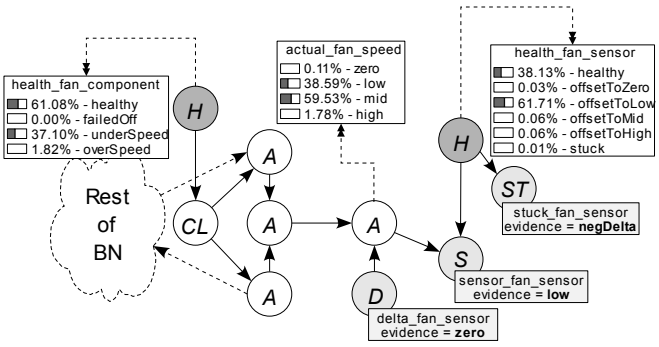

Figure 13: The marginal distributions for health *H* nodes health_fan_component and health_fan_sensor as well as the actual_fan_speed attribute *A* node, when the fan sensor's evidence (sensor reading - state) is changed to low.

Suppose now that the sensor readings for the same fan sensor dip downward so that the discretized state for the sensor *S* node is now *low* (Figure 13). Assuming the evidence clamped to the rest of the Bayesian network is the same as in Figure 12, we see that the most likely value for the sensor's health is now *offsetToLow*, based on the marginal distribution for that node (Figure 13). However, there is still enough evidence to suggest that the sensor's health could be *healthy*, but with a lower probability. Therefore, we say that the sensor's health is *offsetToLow*. A similar logic applies to the fan component's health state as being *healthy*.

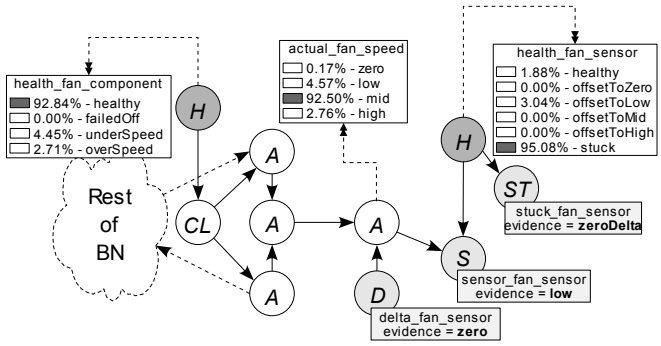

Figure 14: The marginal distributions for health *H* nodes health_fan_component and health_fan_sensor as well as the actual_fan_speed attribute *A* node, when the stuck *ST* node is clamped to the stuck state.

Now we show what happens when ProDiagnose determines that a sensor is *stuck*. In Figure 14, the stuck *ST* node is clamped to *zeroDelta*, the Bayesian network name for a stuck state. Again assuming the evidence in the rest of the Bayesian network is the same as in Figures 12 and 13, we see that the most likely value for the sensor's health is *stuck*, with high probability, based on the marginal distribution (Figure 14).

Next we show another example, this time illustrating a Change node *CH* in a configuration with a battery.

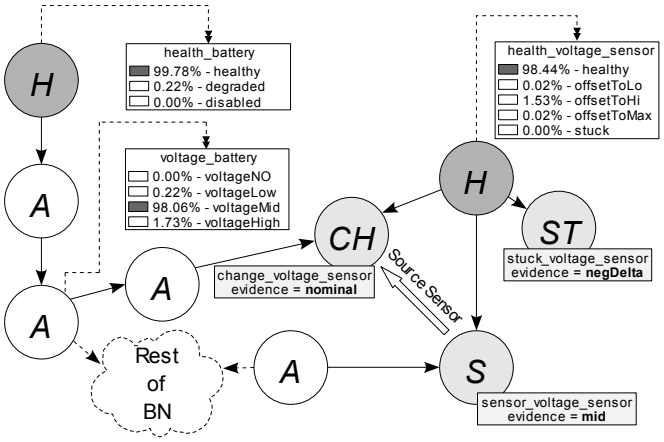

Figure 15: The marginal distributions for health *H* nodes health_battery and health_voltage_sensor as well as the voltage_battery attribute *A* node, for nominal ADAPT battery behavior.

Figure 15 represents the current states of a battery and a voltage sensor downstream from the battery. The battery component is represented by the *H* node and three *A* nodes in the left side of Figure 15, to the left of the 'Rest of BN' cloud. The Change *CH* node derives its state from discretizing the CUSUM from the Sensor *S* node (sensor_voltage_sensor). This relationship is illustrated by the source sensor arrow in Figure 15. The *S* node is considered to be the bounded sensor to the *CH* node. This example starts off in a nominal state, with both the battery and voltage sensor in a healthy state.

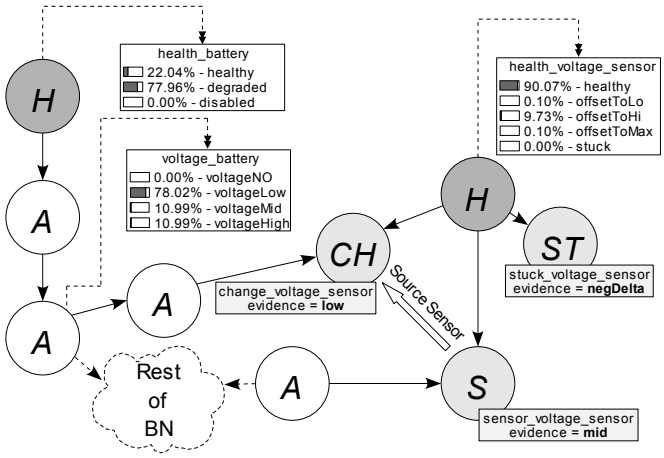

Figure 16: The marginal distributions for health *H* nodes health_battery and health_voltage_sensor as well as the voltage_battery attribute *A* node, when a battery's voltage has dropped enough for the battery to be considered degraded.

Figure 16 illustrates what happens in the BN after the battery's voltage starts to drop slightly. The voltage sensor downstream from the battery (right side of Figure 16) is still showing a state of 'mid' because the drop in voltage is not enough to cross the threshold to a lower voltage state. However the CUSUM from this sensor is showing a decreasing voltage long-term trend (and the *CH* node's state changes from 'nominal' to 'low'), which is interpreted in the BN as a degrading battery. Therefore, the battery's health state changes from 'healthy' to 'degraded'. Notice how the voltage sensor's health is still 'healthy'. The next figure shows why this is important.

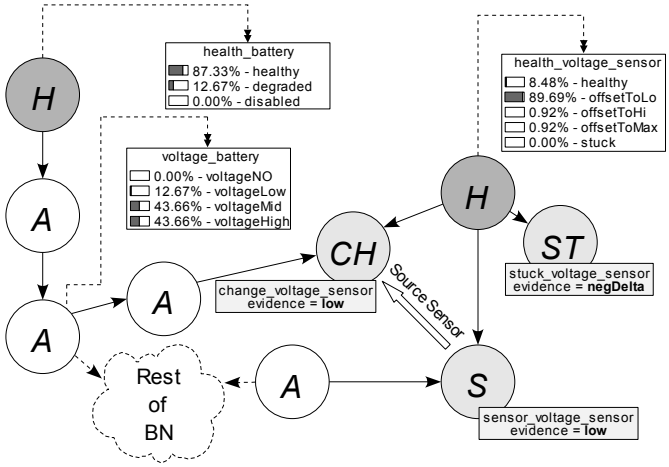

Figure 17: The marginal distributions for health *H* nodes health_battery and health_voltage_sensor as well as the voltage_battery attribute *A* node, when the voltage sensor is offset.

Figure 17 shows what happens if the bounded sensor to a *CH* node is in an unhealthy state. In this example, the sensor becomes offset at a much lower voltage reading than it should be at, and its state changes to 'low' (even though it should be at 'mid'). This offset results in the health of the sensor changing to the 'offsetToLo' state. This offset also

causes a downward trend in the CUSUM being derived from the sensor's readings, and this changes the *CH* node's state from 'nominal' to 'low'. However, because the sensor is deemed to be unhealthy, the *CH* node's state, having been derived from the unhealthy sensor's readings, can no longer be considered accurate. We give equal conditional probability to all states in the *CH* node (Table 4) when the sensor's *H* node is showing an unhealthy state (sensor is unhealthy). This ensures that the battery component (Figure 17) is not affected by any one *CH* node state over another.

7 ELECTRICAL POWER SYSTEM CASE STUDY

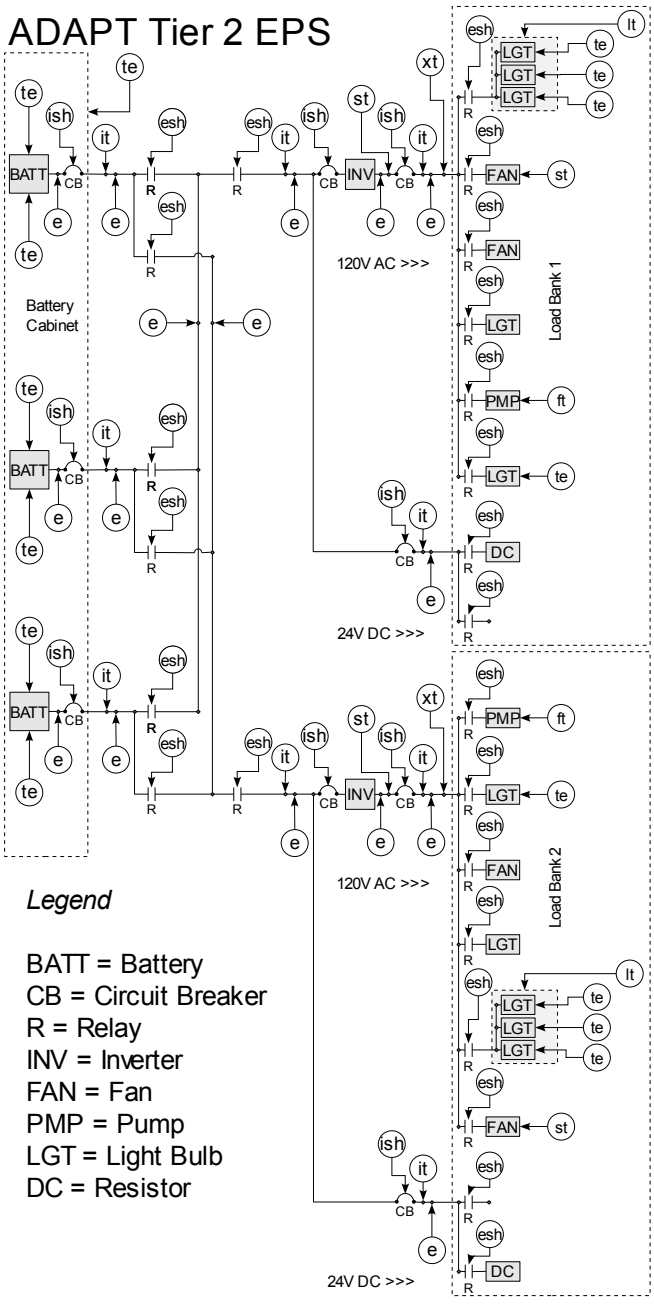

Figure 18: The ADAPT Tier 2 Electrical Power System. Tier 2 represents the full EPS, which we will refer to as simply ADAPT.

7.1 The ADAPT EPS

ADAPT EPS					
			ADAPT	Bayesian Network	
Name	Sym	Description	Qty per EPS	Qty per sensor	
				Nodes	Evidence nodes
DC Current Sensor	it	Measures DC current in amps	7	3	2
AC Current Sensor	it	Measures AC current in amps	2	3	2
DC Voltage Sensor	e	Measures DC voltage in volts	12	3	2
AC Voltage Sensor	e	Measures AC voltage in volts	4	3	2
Circuit Breaker Position Sensor	ish	Senses whether a circuit breaker is opened or closed	9	2	1
Relay Position Sensor	esh	Senses whether a relay is opened or closed	24	2	1
Temperature Sensor	te	Measures temperature in Fahrenheit of batteries, battery cabinet, and light bulbs	15	5	3
Speed Transmitter	st	Measures RPM of the large fans	2	5	3
Phase Angle Transducer	xt	Measures the phase shift in degrees between the sine waves of AC current and voltage	2	6	2
AC Frequency Transmitter	st	Measures the AC frequency in Hertz	2	3	2
Flow Transmitter	ft	Measures the flow rate in gallons per hour through a pump	2	5	3
Light Sensor	lt	Measures the intensity in millivolts of incoming light	2	3	2
		TOTAL	83	43	25

Table 5: ADAPT EPS sensors, with their quantity in the ADAPT Tier 2 EPS. Also listed are the node and evidence node quantities for each sensor.

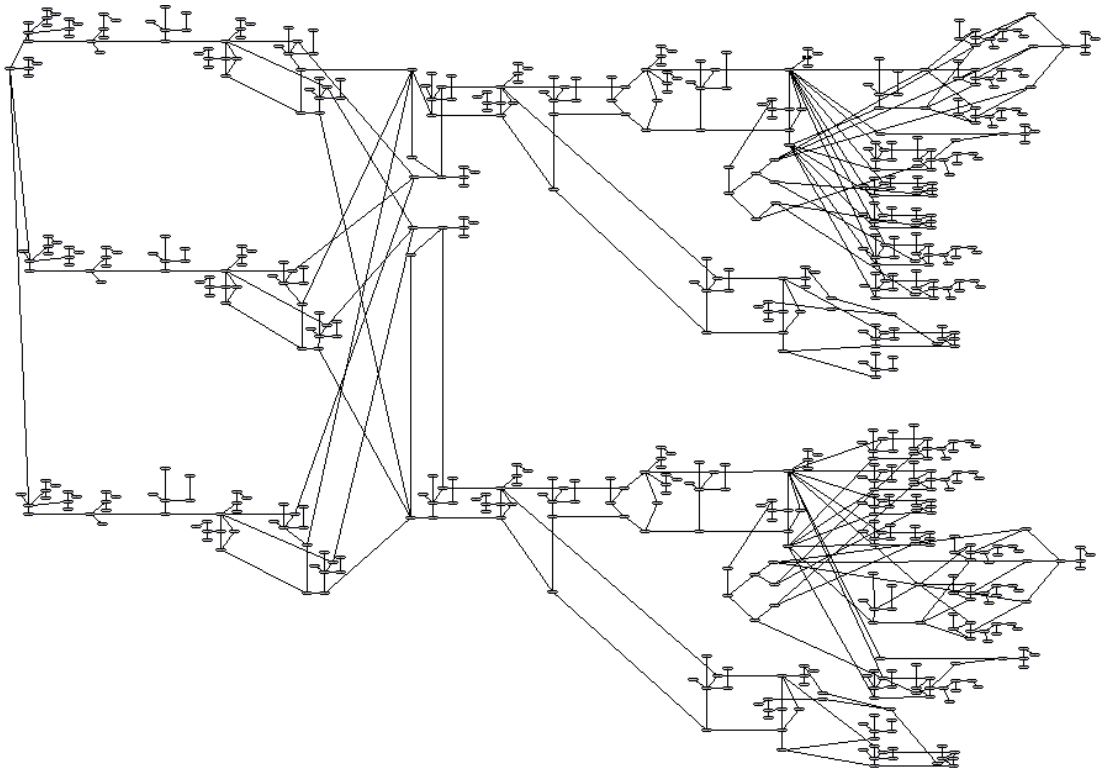

Figure 19: The Bayesian network representation of the ADAPT EPS. The layout visually is similar to the physical system itself.

The Advanced Diagnostics and Prognostics Testbed, or ADAPT, is a real-world Electrical Power System (EPS) located at NASA Ames Research Center. Its physical structure is illustrated in Figure 18. This EPS is similar to electrical power systems found aboard NASA spacecraft and aircraft (Mengshoel et al., 2008). Since Tier 2 represents the ADAPT EPS in its entirety, we will refer to it simply as ADAPT.

ADAPT Tier 2 (ADAPT) consists of 3 batteries connected in parallel through 2 DC → AC inverters to 2 load banks (Poll et al., 2007, see Figure 18). Each load bank has a DC loads section that bypasses the inverter. Reference Table 5 for a breakdown of sensor quantity.

7.2 Bayesian Network for ADAPT

Figure 19 shows the ADAPT EPS as a Bayesian network. It currently employs 671 nodes, 789 edges and has a domain cardinality of [2, 16], with an average cardinality of 2.86. The general structure of the BN model is laid out to visually mimic the structure of the ADAPT EPS (Figure 18).

8 ELECTRICAL POWER SYSTEM EXPERIMENTS

We first describe the framework, developed at NASA Ames Research Center, used in the experimental process for collecting our results.

8.1 The DXC Framework

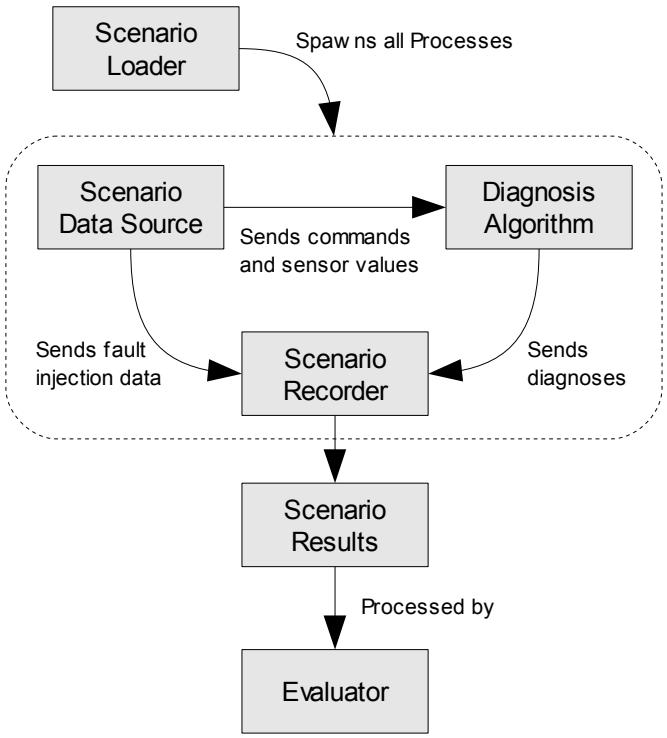

Figure 20: The DXC Framework used for evaluating ProADAPT's experimental results. The Diagnosis Algorithm would be ProADAPT in our case.

The DXC (or DCC as Diagnostic Challenge Competition is sometimes abbreviated) framework provides a method of interfacing ProADAPT with a scenario evaluator using a common protocol (Kurtoglu et al., 2009a; Kurtoglu et al., 2009b; Figure 20).

ProADAPT incorporates the framework's API to interface with the *scenario data source,* which acts as the environment (see Figure 5). It also allows ProADAPT to connect to the *scenario recorder* for diagnostic output. This output *(scenario results)* is then processed using the *evaluator* (Kurtoglu et al., 2009a; Kurtoglu et al., 2009b).

For the experiments in this paper we used all 120 scenarios from the DXC-09 Tier 2 competition set for experimentation. Each scenario simulates an actual fault(s) of components and/or sensors in the ADAPT EPS. These scenarios are either nominal, single, double or triple fault, with various relay and circuit breaker open/close commands (Kurtoglu et al., 2009a; Kurtoglu et al., 2009b). The ADAPT EPS starts in a powered down state, in that all commandable relays are open. Then various relays are closed (and some possibly opened again), depending on the scenario.

These scenarios were run on ProDiagnose using the latest BN model[2]. ProDiagnose also competed in the DXC-09 Competition under the name *ProADAPT*[3], and we will compare our newest results to the ProADAPT results from the competition, along with results from other diagnostic algorithms.

For notational simplicity, in this section we will refer to the version of ProADAPT that competed in DXC-09 as ProADAPT1, and the newest version as ProADAPT2.

8.2 Notation and Definitions

The competition results are based on multiple metrics, which we will now briefly summarize.

A *false positive* refers to detecting a fault when a fault is not present. A *false negative* refers to not detecting a fault when a fault is present. Reporting a fault many seconds after the fault injection occurs is acceptable, so long as the fault is reported before the scenario ends.

Detection accuracy is the percentage of correct fault detections when taking into account the total percentage of false positives and false negatives. A detection accuracy of 100% would imply a 0% false positives and false negatives rate (but may not necessarily imply 0 *classification errors,* see below).

Mean time to detect refers to the time elapsed between specific fault injection and first detection of *a* fault. *Mean time to isolate* is similar to the mean detection time, except that an *isolation* refers to identification of the *correct* fault. Diagnosing the wrong fault will result in an isolation time equal to the difference between the initial fault injection time and the end of scenario time, which is usually a very high number.

Classification errors refer to the number of misdiagnoses made during an entire scenario run (all 120 in this case). Misdiagnoses include both false positives and false negatives. It is possible to have multiple classification errors per scenario. Classification errors that occur *after* a fault injection, such as diagnosis of an incorrect fault, will not result in a false negative or positive, as these types of classification errors still count as a correct *detection* (of a fault). An example is a fault in which a voltage sensor becomes stuck at 0. If the diagnosis is that INV1 (inverter) failed, then we get hit with 2 classification errors, one for not isolating the correct fault (voltage sensor being stuck), and another for isolating a fault that didn't exist (the inverter failing). Another instance in which classification errors can occur is in multiple-fault scenarios, in which all faults are initially isolated correctly, but then one of the faults is retracted. Because the other faults are still being correctly diagnosed, the retracted fault is "forgotten", as the last diagnosis output before the end of the scenario is the one used for accuracy (so if all these faults were retracted at the same time, the last diagnosis would show all correct isolations).

Mean CPU Time is a measure of mean CPU resources used by ProDiagnose over all scenarios run, and *Mean Peak RAM Usage* measures the maximum amount of memory needed by ProDiagnose averaged over all run scenarios.

8.3 Results for the DXC-09 Competition - ProADAPT1

The table on the next page shows the results from the DXC-09 Competition:

[2]The BN file used was the latest as of August 8th, 2009, and is named adapt10f3_v5c.net. Discretization and other relevant information is kept in the file v5cT2.plog.
[3]The BN file used for the DX 09 Competition in this paper is named DXCT2.net (April 6th, 2009). Discretization and other relevant information is kept in the file DXCT2.plog.

DXC-09 ADAPT Industrial Track Tier 2						
	ProDiagnose (ProADAPT1)	**FaultBuster**	**HyDE**	**RODON**	**Stanford**	**Wizards Of Oz**
False Positives	7.32%	81.43%	0.00%	54.17%	32.16%	51.06%
False Negatives	13.92%	24.00%	30.00%	9.72%	5.19%	9.59%
Classification Errors	76	130	121.57	84.01	110.55	159.25
Detection Accuracy	88.33%	42.50%	80.00%	72.50%	85.00%	74.17%
Mean Time to Detect	5973 ms	14099 ms	17610 ms	3490 ms	3946 ms	30742 ms
Mean Time to Isolate	11988 ms	37808 ms	21982 ms	36331 ms	14103 ms	47625 ms
Mean CPU Time	2922 ms	5798 ms	29612 ms	80261 ms	963 ms	23387 ms
Mean Peak RAM Usage	6539 KB	10261 KB	20515 KB	29878 KB	5912 KB	7498 KB

Table 6: Results of the DXC-09 Industrial Track Tier 2 Competition.

Looking at Table 6 from the DXC-09 Competition, ProADAPT1 did very well in all categories. ProADAPT1 had the lowest number of classification errors with 76, along with the mean fastest mean isolation times at just under 12 seconds. It also had the highest detection accuracy at 88.33% (Ricks and Mengshoel, 2009).

ProADAPT1 is characterized by very low CPU and RAM usage, which makes it ideal for systems with tight resource constraints, such as those found on various types of aircraft and spacecraft.

It may seem that an 11 second mean isolation time is high (despite being the fastest time), but this is in large part due to stuck faults, as ProADAPT1 waits to ensure with high confidence that a sensor is indeed stuck before submitting a diagnosis for it. Faults involving components such as fans and pumps usually will have high isolation times also, due to a similar principle of waiting (Ricks and Mengshoel, 2009). In this case, ProADAPT1 waits until the component's sensor readings trip a certain threshold, and the diagnosis is then made based on other node influences within the Bayesian network (The delta *D* node in Figure 9 aids the accuracy of this process). Another cause of higher isolation times are transients caused by the fault injection itself.

Please reference (Kurtoglu et al., 2009a) and (Kurtoglu et al., 2009b) for more information on these metrics.

8.4 Results for updated BN - ProADAPT2

The results in this section use the same scenarios as those from the DXC-09 Competition. ProDiagnose is using an updated BN model and Prolog definitions file, giving ProADAPT2.

ProDiagnose: Latest ADAPT Tier 2 Results Using DXC-09 Industrial Track Tier 2 Scenarios		
	ProADAPT1	**ProADAPT2**
False Positives	7.32%	0.00 %
False Negatives	13.92%	1.25 %
Classification Errors	76	20
Detection Accuracy	88.33%	99.17 %
Mean Time to Detect	5973 ms	2096 ms
Mean Time to Isolate	11988 ms	10961 ms

Table 7: ProDiagnose experimental results using the updated ADAPT BN Model.

A large weakness with the ADAPT model used during the DXC-09 Competition had to do with a restricted physical loads model. Components in the ADAPT testbed such as light bulbs were not accurately represented, and thus did not give accurate states. The new physical loads model fixed these problems, and also added new evidence in the form of a change *CH* node for each bank that monitors change in current leading into the bank itself. This node aids in accurate detection of faults within the banks, especially multiple fault scenarios in which before there sometimes wasn't enough evidence to avoid ambiguity.

This new BN also fixed a flaw in stuck detection having to do with sensors monitoring components that were either off or failed. Fan sensors for example will show 0 RPM consistently when the fan blades are not spinning, but this is not stuck behavior. The new BN model allows for more accurate stuck detection. The older BN often would miss stuck faults completely, decreasing our detection accuracy but also decreasing our average isolation times, due to the missed diagnoses not contributing to our isolation. However, the new BN model combined with generally higher diagnostic accuracy actually resulted in an average isolation time that is a second lower than the DXC-09 results. The DXC-09 results factored in many false negatives

from missed stuck faults that the new model picked up correctly.

The new BN model also incorporates a more generalized XT sensor model. The phase angle measured using these sensors are not always behaviorally consistent across scenarios, and thus this sensor was at times coming up incorrectly as faulty, increasing our false positive rate. This fix (combined with the new loads model) decreased our false positive rate to 0%. This helped push our detection accuracy past 99%.

Our mean detection times dropped by over 50%, in part due to the wider range of faults ProADAPT2 was now able to properly detect. The component faults in the load banks can usually be properly detected within 500ms (faster if no transients are present). Our >99% detection accuracy is a testament to the new loads model and updated stuck detection (Table 7).

8.5 Raw Detection Performance (Inference)

Stuck detection and other factors that pop up during fault diagnoses can often lead to longer detection and isolation times. In disregarding these factors, we can get a good indication of ProDiagnose's inference times.

We ran ProDiagnose through Tier 2 scenarios from the DXC-09 Competition, including only those that involve faults that a DA can catch immediately without any other factors involved. ProDiagnose will calculate and report the same fault each time sensor readings come in after the fault injection (not including fault withholding due to parameters like t_{CO}, and providing all evidence remains the same). Because of this, we were able to take the detection times as being the diagnosis after the *second* sensor sample from the fault injection.

Raw Fault Detection Times	
Fault Type	**Isolation Time**
ISH236=stuck	1 ms
ESH272=stuck	1 ms
ST516=offset	1 ms
ESH284=stuck	1 ms
E281=Offset	1 ms
Average	1 ms

Table 8: Fault detection times for faults that are not influenced by stuck detection.

ProDiagnose consistently had detection times at 1 ms. This though is the smallest unit of time that can be measured before Java starts to break down in accuracy. Also ACE inference times increase as the BN model increases in size. However, when factoring in any CPU time used by all ProDiagnose functions outside of the inference engine, it is very probable that ProDiagnose's inference times are in fact < 1 ms on average.

9 CONCLUSION AND FUTURE WORK

There is need for methods that bridge the gap between complex systems, including electrical power systems, that are hybrid and may also exhibit other challenging behaviors. Most existing diagnostic technologies typically have a discrete or continuous foundation, and diagnostics in a hybrid, complex setting is an important topic for on-going research. In this paper, we have presented methods for hybrid diagnosis by means of discrete probabilistic models (Bayesian networks and arithmetic circuits), and specifically discussed novel techniques for handling continuous stuck faults and continuous offset faults. These techniques are embedded in the ProDiagnose algorithm.

In experiments with the ADAPT EPS, ProDiagnose turns out to compute highly accurate diagnoses by means of probabilistic models. It is characterized by quick detection and isolation times, with a high degree of accuracy for detecting faults. Part of this success was due to the presence of certain BN nodes (Delta nodes, Stuck nodes, and Change nodes) that address the challenges discussed above and which are additions compared to an earlier version of the ADAPT BN (Mengshoel et al. 2008). These novel BN nodes are coupled with dynamic processing in ProDiagnose to calculate their discrete states from continuous sensor measurements. An improved physical loads model in the ADAPT BN also helped to greatly improve performance and accuracy in the most recent results reported here.

Future work includes adding dynamic Bayesian network (DBN) capabilities to ProDiagnose in order to improve detection accuracy, especially for fault types not discussed here. Much of this research will be focused on implementing reliable DBN models from the static models currently used, as well as possible computational challenges associated with DBNs.

ACKNOWLEDGMENTS

We would like to thank Scott Poll, David Garcia, David Nishikawa and numerous others at the NASA Ames Research Center for generating the ADAPT data for our experiments, and for helping in many other ways. This material is based upon work supported by NASA under awards NCC2-1426 and NNA07BB97C.

REFERENCES

(Bickmore, 1992) T. W. Bickmore. A Probabilistic Approach to Sensor Data Validation, In *Proceedings of the 28th Joint Propulsion Conference and Exhibit*, (Nashville, TN), 1992.

(Button and Chicatelli, 2005) R. M. Button and A. Chicatelli. Electrical Power System Health Management. In *Proceedings of the 1st International Forum on Integrated System Health Engineering and Management in Aerospace*, (Napa, CA), 2005.

(Bunus et al., 2009) Peter Bunus, Olle Isaksson, Beate Frey, Burkhard Münker. RODON - A Model-Based Diagnosis Approach for the DX Diagnostic Competition. In *Proceedings of 20th International Workshop on Principles of Diagnosis (DX-09)*, (Stockholm, SE), pp. 423-430, 2009.

(Chavira and Darwiche, 2007) M. Chavira and A. Darwiche. Compiling Bayesian Networks using Variable Elimination. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, (Hyderabad, India), pp. 2443-2449, 2007.

(Chien et al., 2002) C. Chien, S. Chen, and Y. Lin. Using Bayesian Networks for Fault Location on Distribution Feeder. *IEEE Transactions on Power Delivery,* vol. 17**,** pp. 785-793, 2002.

(Daigle et al., 2008) M. Daigle, X. Koutsoukos, and G. Biswas. An Integrated Approach to Parametric and Discrete Fault Diagnosis in Hybrid Systems. In *Proceedings of 11th International Workshop on Hybrid Systems: Computation and Control (HSCC-08)*, (St. Louis, MO), pp. 614–617, 2008.

(Darwiche, 2000) A. Darwiche. Model-Based Diagnosis under Real-World Constraints. *AI Magazine*, vol. 21, no. 2**,** pp. 57-73, 2000.

(Darwiche, 2003) A. Darwiche. A Differential Approach to Inference in Bayesian Networks. *Journal of the ACM*, vol. 50, no. 3**,** pp. 280-305, 2003.

(Gorinevsky et al., 2009) D. Gorinevsky, S. Boyd, and S. Poll. Estimation of Faults in DC Electrical Power System. In *Proceedings of the American Control Conference*, 2009.

(de Kleer and Williams, 1987) J. de Kleer and B. C. Williams. Diagnosing Multiple Faults, *Artificial Intelligence,* 32(1), pp. 97-130, 1987.

(Koller and Lerner, 2000) D. Koller and U. Lerner. Sampling in Factored Dynamic Systems. In *Sequential Monte Carlo Methods in Practice,* 2000.

(Kurtoglu et al., 2008) T. Kurtoglu, O. J. Mengshoel, and S. Poll. A framework for systematic benchmarking of monitoring and diagnostic systems. In *Annual Conference of the Prognostics and Health Management Society (PHM-08)*, 2008.

(Kurtoglu et al., 2009a) T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, and A. Feldman. First International Diagnosis Competition – DXC'09. In *Proceedings of 20th International Workshop on Principles of Diagnosis (DX-09)*, (Stockholm, SE), pp. 383–396, 2009.

(Kurtoglu et al., 2009b) T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund and A. Feldman. Towards a Framework for Evaluating and Comparing Diagnosis Algorithms. In *Proceedings of the 20th International Workshop on Principles of Diagnosis (DX-09)*, (Stockholm, SE), pp. 373–382, 2009.

(Lauritzen and Spiegelhalter, 1988) S. Lauritzen and D. J. Spiegelhalter. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems (with Discussion). *Journal of the Royal Statistical Society series B*, vol. 50, no. 2, pages 157-224, 1988.

(Lerner et al., 2000) U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *Proceedings of The Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pp. 531–537, 2000.

(Liu and Zhang, 2002) E. Liu and D. Zhang. Diagnosis of Component Failures in Space Shuttle Main Engines using Bayesian Belief Networks: A Feasibility Study. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-02),* (Washington D.C.), 2002.

(Mengshoel et al., 2006) O. J. Mengshoel, D. C. Wilkins and D. Roth. Controlled Generation of Hard and Easy Bayesian Networks: Impact on Maximal Clique Tree in Tree Clustering. *Artificial Intelligence (170),* pp. 1137-1174, 2006.

(Mengshoel, 2007) O. J. Mengshoel. Designing Resource-Bounded Reasoners Using Bayesian Networks: System Health Monitoring and Diagnosis. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, (Nashville, TN), pp. 330-337, 2007.

(Mengshoel et al., 2008) O. J. Mengshoel, A. Darwiche, K. Cascio, M. Chavira, S. Poll, and S. Uckun. Diagnosing Faults in Electrical Power Systems of Spacecraft and Aircraft. In *Proceedings of the Twentieth Innovative Applications of Artificial Intelligence Conference (IAAI-08)*, (Chicago, IL), pp. 1699-1705, 2008.

(Mengshoel et al., 2009) O. J. Mengshoel, M. Chavira, K. Cascio, S. Poll, A. Darwiche, and S. Uckun. Probabilistic Model-Based Diagnosis: An Electrical Power System Case Study. Accepted for publication in *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans,* 2009.

(Narasimhan and Biswas 2007) S. Narasimhan and G. Biswas. Model-Based Diagnosis of Hybrid Systems. *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans,* 37(3): pp. 348-361, 2007.

(Olesen, 1993) K. G. Olesen. Causal Probabilistic Networks with Both Discrete and Continuous Variables. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 15(3), pp. 275-279, 1993.

(Pearl, 1988) J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.

(Poll et al., 2007) S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. J. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, A. Sweet, S. Yentus, I. Roychoudhury, M. Daigle, G. Biswas, and X. Koutsoukos. Advanced Diagnostics and Prognostics Testbed. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, (Nashville, TN), pp. 178-185, 2007.

(Ricks and Mengshoel, 2009) B. Ricks and O. J. Mengshoel. The diagnostic challenge competition: Probabilistic Techniques for Fault Diagnosis in Electrical Power Systems. In *Proceedings of 20th International Workshop on Principles of Diagnosis (DX-09),* (Stockholm, SE), pp. 415–422, 2009.

(Yongli et al., 2006) Z. Yongli, H. Limin, and L. Jinling. Bayesian Network-Based Approach for Power System

Fault Diagnosis. *IEEE Transactions on Power Delivery*, vol. 21**,** pp. 634-639, 2006.

Brian Ricks is an undergraduate student at the University of Texas at Dallas. He worked for the NASA Ames Research Center, Intelligent Systems Division, as an intern with the Universities Space Research Program. He will graduate in Spring of 2010 with a BS in Computer Science from the University of Texas at Dallas. Mr. Ricks performed part of the research reported here during his internship, under the leadership of Dr. Ole J. Mengshoel.

Ole Mengshoel is a Senior Scientist with Carnegie Mellon Silicon Valley at the NASA Ames Research Center, Intelligent Systems Division. Dr. Mengshoel has managed and provided hands-on leadership in a wide range of research and development projects. He has successfully developed technical results and software that have or are being matured and transitioned into the aerospace, defense, finance, education, electronic commerce, and manufacturing sectors. His current research focuses on reasoning, diagnosis, decision support, reasoning, and machine learning under uncertainty - often using Bayesian networks – with aerospace applications of interest to NASA. He holds a Ph.D. in Computer Science from the University of Illinois, Urbana-Champaign. His undergraduate degree is in Computer Science from the Norwegian Institute of Technology, Norway (now NTNU).